PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 11/20, 11/14

A1

(11) International Publication Number: WO 92/18931

(43) International Publication Date: 29 October 1992 (29.10.92)

(21) International Application Number: PCT/US92/03001

(22) International Filing Date: 14 April 1992 (14.04.92)

(30) Priority data:
690,066 23 April 1991 (23.04.91) US

(71) Applicant: EASTMAN KODAK COMPANY [US/US]; 343 State Street, Rochester, NY 14650 (US).

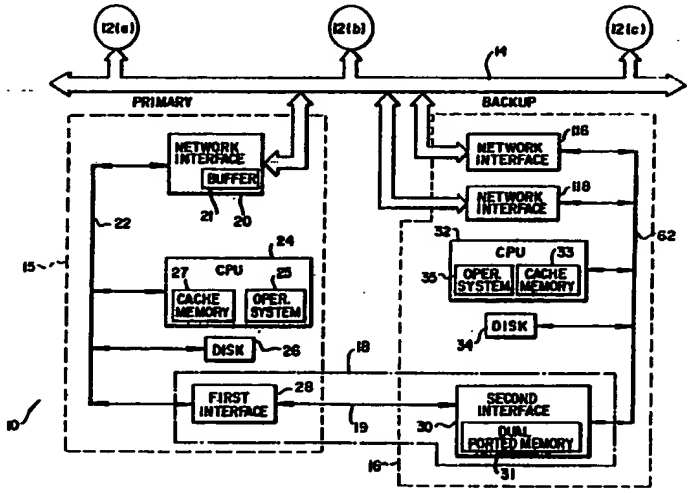
(72) Inventors: VINTHER, Gordon ; 22 Jersey Street, Pepperell, MA 01463 (US). McGRATH, James, W. ; 108 Kinnaird Street, Cambridge, MA 02139 (US).

(74) Agent: DUDLEY, Mark, Z.; 343 State Street, Rochester, NY 14650-2201 (US).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, LU (European patent), MC (European patent), NL (European patent), SE (European patent).

Published
With international search report.

(54) Title: FAULT TOLERANT NETWORK FILE SYSTEM



(57) Abstract

A fault tolerant network fileserver system includes a plurality of nodes connected to a network communication link. A primary fileserver node stores files from a plurality of the nodes and a backup fileserver node stores copies of files from the primary fileserver. In an improved fileserver system, the primary and backup fileservers are connected to a dual ported memory for communicating information between the fileservers. The primary fileserver writes data files to the dual ported memory and interrupts a processor within the backup fileserver to notify it that the dual ported memory contains data. In response to the interrupt, the processor within the backup fileserver reads the data from the dual ported memory and writes it to a storage device within the backup fileserver. In a similar manner, the dual ported memory is used for passing control messages between the primary and backup fileservers. The dual ported memory includes semaphore locations for arbitrating between competing requests by the backup and primary fileservers for access to the same location in the dual ported memory.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	ML	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NL	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		
ES	Spain	MG	Madagascar		

FAULT TOLERANT NETWORK FILE SYSTEM

BACKGROUND OF THE INVENTION

This invention relates to a fault tolerant network file system having a primary fileserver and a backup fileserver which mirrors the primary. When the primary fails, the backup assumes the role of the primary on the network in a manner transparent to users whose files are stored on the primary.

A network generally includes a group of nodes which communicate with each other over a high speed communications link. Examples of nodes include single user personal computers and workstations, multiple user computers, and peripheral devices such as image scanners, printers and display devices.

Many networks include a fileserver node which operates as a central storage facility for data and software used by many nodes on the network. The fileserver includes a disk storage device, for storing the data and software, and a central processing unit (CPU) for controlling the fileserver. Files arriving at the fileserver over the communication link are typically first stored in a cache memory within the central processing unit and later copied to the disk storage device for permanent storage.

If the fileserver fails before the files in cache memory are copied to the disk, the files may be irretrievably lost. Accordingly, a client node sending certain critical files to the fileserver may instruct the fileserver to immediately write specified files to disk, thereby reducing the likelihood that the specified files will be lost in the event of such a failure.

Since many nodes rely on the fileserver, many users will be affected if the fileserver fails. For example, even if all files are safely stored on the primary fileserver's disk, a temporary failure of the

primary fileserver will inconvenience many users since their data files are unavailable until the fileserver is restored to service. Networks which require a high degree of availability typically employ techniques to hedge against such failure. For example, a backup fileserver may be used to maintain a copy of the primary's files. Upon failure of the primary, an exact copy of the primary's files can be automatically accessed through the backup fileserver.

One object of the invention is to provide a backup fileserver which promptly mirrors each change of the primary's files. Thus, when the primary fails, the files of the backup fileserver match those of even the most recently added or modified files of the primary. A further object of the invention is to achieve this mirroring at high speed and with little computational cost. Yet another object is to provide the elements for switching between the primary fileserver to the backup fileserver in a manner transparent to the users.

SUMMARY OF THE INVENTION

The invention relates to an improved fault tolerant network fileserver system. The network fileserver system includes a network communication link connected to a plurality of nodes. A primary fileserver and a backup fileserver are also connected to the network communication link for storing files from the nodes.

In the improved fileserver system, the primary fileserver includes a primary computer processor, a primary storage disk, a first network interface connected to the network communication link, and a first independent interface connected to the backup fileserver. The first independent interface is responsive to commands from the primary processor for communicating information to the backup fileserver.

The backup fileserver includes a backup computer processor, a backup storage disk, a backup network interface connected to the network communication link, and a second independent interface connected to the first independent interface. The second independent interface includes a dual ported memory. It receives information from the primary fileserver and stores the information in the dual ported memory. In response to commands from the back-up processor, the second independent interface provides information from the dual ported memory.

In preferred embodiments, the second independent interface includes a means for interrupting the backup computer processor to notify it that the dual ported memory contains information received from the primary fileserver. A portion of the dual ported memory is arranged as a group of data memory blocks, each for storing data to be copied to the backup storage disk. Other portions store an entry pointer identifying a next available data block, and a removal pointer identifying a next data block to be emptied.

Other portions of the dual ported memory may be arranged as a first control message block, for storing control messages from the primary fileserver to the backup fileserver, and a second control message block, for storing control messages from the backup fileserver to the primary fileserver.

The second independent interface includes at least one common register accessible to both the primary and backup processors for indicating the entry and removal of data blocks from the data block portion. For example, a preferred embodiment includes a count register. The second interface includes a means responsive to signals from the first independent interface for modifying the contents of the count register to indicate the entry of data into a memory

block. The second independent interface is further responsive to signals from the backup computer processor for modifying the count register to indicate the removal of data from a memory block.

The dual ported memory includes a semaphore register for arbitrating access to the count register. The semaphore register has a first and a second state of operation. When read by a processor while in the first state, the semaphore register provides a register available code and automatically enters the second state. When read by a processor while in the second state, the semaphore register provides a register unavailable code and remains in the second state. When written to by a processor while in the second state, the semaphore register returns to the first state.

In the preferred embodiment, the primary fileserver further includes an improved Unix operating system for controlling the copying of files received by the first network interface to the primary storage disk and to the dual ported memory. Conventional Unix operating systems receive both secure disk write instructions and unsecure disk write instructions directing the operating system to write specified files to the primary storage disk. In response to an unsecure write instruction, an unsecure write procedure writes to the primary storage disk using an efficiency algorithm which temporarily defers copying the files to disk. In response to a secure write instruction, a secure procedure writes files to disk promptly without the benefit of the efficiency algorithms. The improved operating system responds to both secure and unsecure instructions by promptly writing the specified files to the dual ported memory, and subsequently writing the specified files to the primary disk storage device using the unsecure write procedure.

Other objects, features and advantages of the invention are apparent from the following description

of a preferred embodiment taken together with the drawings.

Brief Description of the Drawings

Figure 1 is a block diagram of a computer network having a primary fileserver and a backup fileserver.

Figure 2 is a block diagram of a pair of interface boards for connecting the primary fileserver to the backup fileserver.

Figure 3 is a diagram of several registers on the interface boards shown in Figure 2.

Figure 4 is a diagram showing the organization of a dual ported memory according to a preferred embodiment of the invention.

Figures 5(a) and 5(b) are a flow chart of a method for copying new files of the primary fileserver to the dual ported memory.

Figures 6(a) and 6(b) are a flow chart of a method for copying data from the dual ported memory to a backup fileserver.

Figures 7(a) through 7(d) are a flow chart of a procedure by which a backup fileserver temporarily replaces the primary fileserver.

Figures 8(a) and 8(b) are a flow chart of a procedure by which a failed primary fileserver resumes its role on the network after recovering from a failure.

Description of the Preferred Embodiment

Referring to Fig. 1, a network 10 includes a plurality of nodes 12, each for performing specific tasks in the design and production of publications such as newspapers and magazines. For example, node 12(a) could be a computer workstation which allows a journalist to draft an article to be included in the publication; node 12(b) could be a scanner for digitizing an image, such as a photograph, to be

printed with the article prepared at node 12(a); and node 12(c) could be a computer workstation which allows a user to lay out the entire publication by selecting and arranging articles, images and advertisements prepared by other nodes on the network.

The network may include other nodes for producing hard copies of the publication. For example, printer nodes prepare hard copies of images called "proofs". Other nodes prepare printing plates used in high volume printing of the publication.

Each node 12 communicates with other nodes via a high speed communication link 14. For example, in networks conforming to the Ethernet protocol, link 14 is a high speed serial communication channel over which a node can broadcast messages to one or more other nodes. Many nodes include disks for storage of information used locally on the node. For example, workstations 12(a) and 12(c) typically include disks for storing software used in performing their respective tasks, e.g., text editing and document layout.

However, many nodes are diskless and therefore require access to a central fileserver 15 for storing information and accessing software. Further, even nodes having disks often store data on the central fileserver. Requiring nodes to store data files in a central fileserver provides several advantages including the centralized control of a common file system which is accessible to many nodes on the network.

Failure of the fileserver node is catastrophic since the fileserver contains information needed by a large number of nodes. Accordingly, a backup fileserver 16 is connected to the primary fileserver through a parallel port 18 to maintain a copy of all of the primary's files. Upon failure of the primary, the backup automatically assumes the role

of the primary in a manner transparent to the other nodes on the network. More specifically, other nodes may continue to access the central filesystem without any special instruction from the users.

The primary fileserver includes a first network interface 20 for receiving messages (e.g., files) from communications link 14 and storing them in a buffer memory 21. A central processing unit CPU 24 responds to instructions from operating system software 25 to transfer the buffer contents over a parallel bus 22, through a first interface 28, across a cable 19 to a second interface 30 within backup fileserver 16. Second interface 30 includes the dual ported memory 31 which temporarily stores the buffer data.

After storing data in memory 31, CPU 24 instructs interface 30 to interrupt a CPU 32 within backup fileserver 16, thereby notifying CPU 32 that memory 31 includes data. In response to the interrupt, CPU 32 moves the buffer data contents of dual ported memory 31 to a cache memory 33. An operating system 35 later instructs CPU 32 to copy the data from cache 33 to the backup disk 34, thereby providing disk 34 with a copy of each file arriving from the network.

After CPU 24 has copied the buffer data to the dual ported memory, the operating system 25 instructs CPU 24 to move the data to a data cache memory 27. Operating system 25 next instructs CPU 24 to copy the contents of cache memory 27 to a primary disk 26. As explained below, the operating system provides three types of write procedures for copying the files from cache 27 to disk 26.

In the preferred embodiment, operating system 25 is a Unix operating system, modified to include code for interacting with backup fileserver 16 to copy files from buffer memory 21 to dual ported memory 31. Conventional Unix operating systems include filesystem code for maintaining a system of data files on a disk.

The filesystem code includes several types of write procedures for copying files from cache 27 to disk. For files created locally on the primary fileserver (i.e., files which do not arrive over communication link 14), the filesystem code typically employs a delayed write procedure. The delayed write procedure implements efficiency algorithms which manage the writing of data to disk to avoid unnecessary delays. For example, the delayed write procedure searches the cache for files which are assigned to the same disk "cylinder". It then instructs the CPU to initialize a direct memory controller (DMA) to successively transfer the selected files to disk. After this initialization is complete, the CPU returns to other tasks while the DMA controller attends to copying the selected data files to disk. Since the files are stored in the same disk cylinder, a time consuming "cylinder seek" operation is avoided by writing these files successively.

For files arriving over link 14, a "synchronous" write procedure is typically used. The synchronous write procedure promptly writes the file to disk without employing the efficiency algorithms described above. By promptly removing the file from the volatile cache, the synchronous write procedure reduces the likelihood that the file contents will be lost in the event of a failure of the fileserver. However, this procedure degrades disk performance by preventing the efficiency algorithms from reducing the number of time consuming cylinder seek operations.

Further, the client server may also operate in a synchronized fashion, waiting for the fileserver to confirm that the file has been written to disk before proceeding to execute the application program which prompted the write to the filesystem. Such "remote synchronous" writes provides a high degree of security against a failure of the fileserver since the

application program on the client node cannot proceed until the data is safely stored on disk. However, this procedure degrades performance of the client node since the application program is stalled for a long period of time waiting for the relatively slow disk write procedure to complete and for the acknowledge message to arrive over the relatively slow network.

Of the above types of disk writes, the relatively unsecure delayed disk write procedure is the least taxing on the system performance. However, this procedure risks the loss of crucial data. In the event of a failure of the fileserver, contents of the volatile cache memory are destroyed. Accordingly, delayed writes are typically used for relatively unimportant data while synchronous and asynchronous writes are used for relatively critical data.

Backup fileserver 16 provides a high degree of security against a failure without the need for the costly synchronous disk writes. More specifically, each file is initially copied to the dual ported memory 31 which is powered and controlled by the backup fileserver. Since the backup fileserver is thus independent of the primary, there is no need to immediately copy the volatile cache memory 27 to the primary's nonvolatile disk 26. Accordingly, operating system 25 is a modified Unix operating system wherein all synchronous disk writes are converted to delayed writes. By eliminating synchronous writes, the performance of the fileserver is dramatically improved since the efficiency algorithms coordinate all disk writes to optimize disk performance. Further, client nodes running application programs which call for many remote synchronous writes (such as document processing applications of the preferred embodiment) receive an acknowledgment from the fileserver as soon as the file has been written to the relatively fast dual ported memory. Thus, the performance of the client node is

dramatically improved since it need no longer wait for the slow disk write procedure.

OPERATION OF DUAL PORTED MEMORY

Referring to Fig. 2, dual ported memory 31 includes a nineteen bit address bus ADDR which is accessible to both CPU 24 and CPU 32. To access a given location in memory 31, CPU 24 first loads a pair of address registers 40, 42 with the address of the location to be accessed. As shown in Fig. 3, register 42 contains the low order sixteen bits of the address (i.e., bits A0 - A15) and register 40 contains the upper three bits of the address (bits A16 - A18). When CPU 24 performs a read or write cycle directed to memory 31, the contents of registers 40, 42 are automatically applied to the address bus ADDR, thereby pointing to a specific location within memory 31.

To load address register 40, CPU 24 performs a write cycle to a predetermined address assigned to the register. An address decoder 44, within interface 28, decodes the address from parallel bus 22. Upon recognizing the predetermined register address, decoder 44 asserts a pair of encoded control signals "REG" which indicate that CPU 24 is requesting access to register 40. A second address decoder 46, within interface 30, decodes the control signals and asserts a register enable signal R1 which selects register 40. The read/write control signal from bus 22, which indicates that a write is being performed, is forwarded by address decoder 44 to interface 30. Decoder 46 receives the forwarded R/W signal and provides a corresponding buffered signal "R/W1" to register 40. Since the buffered read/write signal R/W1 indicates that a write is being performed, the activation of register enable signal R1 causes register 40 to load the data from data bus DB into the register cells.

The data on bus DB is provided by CPU 24 through transceivers 50, 52. More specifically,

address decoder 44 monitors the read/write control signal from bus 22. Upon recognizing a write cycle to a location on interface 30, decoder 44 enables transceiver 50 to drive data from bus 22 across cable 19. Similarly, when decoder 46 recognizes an access to interface 30, it enables transceiver 52 to forward the data from cable 19 to data bus DB.

In the same manner, CPU 24 loads register 42 by performing a write to an address assigned to register 42. In response to this write cycle, decoder 46 asserts a second register enable signal R2 causing the data from bus DB to be loaded into register 42.

Once registers 40, 42 are initialized with the appropriate address, CPU 24 writes data to the addressed location in memory 31 by performing a write cycle to a predetermined address assigned to memory 31. In response, decoder 46 asserts memory register signal R3 which instructs registers 40, 42 to assert the stored address on the memory address bus ADDR. The memory register signal R3 is further provided to a multiplexer 54 which in response, applies a memory access signal "MEM-Select" to memory 31. The buffered read/write control signal R/W1 is also applied to multiplexer 54 which in response asserts a memory read/write control signal "MEM-R/W". Since MEM-R/W indicates that the present cycle is a write cycle, memory select signal "MEM-Select" instructs memory 31 to load the data D0-D16 (see Fig. 3) from data bus DB to the location provided on the address bus ADDR. Thus, the predetermined address assigned to memory 31 behaves as a sixteen bit register 43 whose contents are determined by the contents of the memory location pointed to by address registers 40, 42.

Referring to Fig. 3, register 40 is a sixteen bit register. However, only five bits in the register are used. As explained above, the low order three bits store the high order address bits (A16 - A18). Bit

nine is a write increment bit "WI1". If this bit is set, the address in registers 40, 42 is incremented each time CPU 24 writes to a location in memory 31. Thus, CPU 24 can write to a block of locations in memory 31 by loading registers 40, 42 with the base address of the block and setting the write increment bit. CPU 24 then simply repeatedly writes data to the address dedicated to memory 31. Registers 40, 42 increment the memory address with each write, thereby loading successive locations in memory 31. CPU 24 can similarly read a block of memory by setting read increment bit RI1 in register 40 and successively reading from memory 31.

CPU 32 reads and writes data from memory 31 in an analogous manner. For example, to read a block of data from memory 31, CPU 32 loads a pair of address registers 56, 58 with the address of the first location to be read, and sets read increment bit RI2 in register 56. It next reads data from the predetermined address dedicated to memory 31. An address decoder 60 decodes the address from bus 62 and upon recognizing the address of memory 31, asserts a memory access signal R7 instructing address registers 56, 58 to assert the stored address on address bus ADDR. Access signal R7 is also applied to multiplexer 54 which responds by asserting memory access signal "MEM-Select". In response to access signal MEM-Select, memory 31 provides data D0-D15 from the memory location identified by the address on bus ADDR to data bus DB. Thus, from the perspective of CPU 32, the predetermined address assigned to memory 31 behaves as a sixteen bit register 43 whose contents are determined by the contents of the memory location pointed to by address registers .

Upon recognizing a read cycle from CPU 32 directed to a location on interface 30, decoder 60 asserts an "enable" signal causing a data transceiver

64 to assert the contents of data bus DB onto bus 62, thereby providing CPU 32 with the desired data. When the read increment bit is set, registers 56, 58 automatically increment the stored address. Thus, CPU 32 reads the next location in memory 31 by again reading from the address assigned to memory 31.

Referring to Figs. 4, 5(a), 5(b), and 6 the following describes in more detail the software procedure for moving data from the primary fileserver 15 to the backup fileserver 16.

ORGANIZATION OF DUAL PORTED MEMORY

Fig. 4 illustrates the memory map for dual ported memory 31 of the preferred embodiment. Memory 31 includes a semaphore block 70 at the beginning of memory (i.e. the first 512 locations) which contain semaphores used in controlling access to certain registers to be described below. The semaphore block is followed by a control register block 72 containing various registers used by CPU 24 and CPU 32 in passing data and control messages as described below.

Control register block 72 is followed by a pair of control message blocks 74, 76. The first control message block 74 is used by CPU 24 to send control messages to CPU 32. Each message is defined by three words 75. To send a message, CPU 24 writes the corresponding three word code to a first container 78 within the message block 74. Subsequent messages are written to adjacent containers.

CPU 32 removes the messages on a first-in-first-out basis. By the time CPU 24 loads the last container 80 in the block, CPU 32 has already emptied the first. Thus, after loading the last container, CPU 24 returns to the first container. The control message block 74 thus operates as a circular buffer.

Control message block 76 is used in the same manner to transfer control messages from CPU 32 to CPU

24. CPU 32 loads messages into the circular buffer and CPU 24 removes them on a first-in-first-out basis.

Control message blocks 74, 76 are followed by a group of data memory blocks 82. The data memory blocks are used to transmit data between CPU 24 and CPU 32 in the same manner that control message blocks 74, 76 are used to transmit control messages. More specifically, CPU 24 loads a first block of data into a first memory block 84. As each new block of data arrives, CPU 24 loads the data into the next memory block.

As CPU 24 loads data blocks into memory, CPU 32 removes them on a first-in-first-out-basis. By the time CPU 24 loads the last memory block 86, CPU 32 has already emptied the first memory block 84. Thus, once CPU 24 loads the last memory block 86, it returns to the first memory block 84. Memory blocks 72 therefore operate as a circular buffer.

OPERATION OF CIRCULAR BUFFERS WITHIN DUAL PORTED MEMORY

Referring to Figs. 5(a) and 5(b), the operation of the circular buffers are now described in more detail, using as an example the transfer of data blocks.

Control register block 72 includes a next entry pointer 88 which points to the next available data block in memory 31. It also includes a removal pointer 90, which points to the next data block to be emptied, and a count register 92 which specifies the number of data blocks currently stored in memory 31.

When CPU 24 desires to load a block of data to memory 31, it first reads count register 92 to determine whether the circular buffer is full. (Step 210). CPU 32 typically removes blocks quickly enough that the circular data buffer should never become full. However, if the buffer becomes full, CPU 24 repeatedly reads the count register 92 until CPU 32 frees a block and decrements the count. (Step 212).

If the buffer is not full, CPU 24 reads the entry pointer 88 to determine the location of the next available data block. (Step 214). It then initializes registers 40, 42 with the address of the first location in the selected block and sets the write increment bit WI in register 40. (Step 216). CPUÿ24 then loads data into the selected memory block by performing successive writes to memory 31. (Step 218). When the block of data is loaded, CPUÿ24 updates the entry pointer 88 to point to the next available block. (Step 220).

Finally, CPU 24 must increment the count to indicate that a new block has been added. However, CPU 32 may also attempt to modify the count at the same time, i.e., to decrement the count to reflect the removal of a data block. For example, assume both CPU 24 and CPU 32 read the current value of the count which is five. CPU 24, having just loaded a new data block, seeks to increment the count to six. CPU 32, having just removed a block, seeks to decrement the count to four. If no mechanism is provided to synchronize CPUÿ24 with CPUÿ32, CPUÿ24 may write a six to the count register 92 and CPUÿ32 will overwrite this value with a four. Yet the count should remain at five since five blocks remain in memory 31.

To avoid this type of error, a count semaphore word 96 is stored in semaphore blockÿ70. (Fig. 4). Before CPUÿ24 reads the count register 92, it first reads count semaphore 96. (Step 222). If the semaphore is zero, (Step 224) CPUÿ24 assumes it has control over the countÿregister.

When CPU 24 reads a zero from semaphore 96, memoryÿ31 automatically sets the semaphore to one, thereby indicating to CPUÿ32 that countÿregister 92 is under the exclusive control of CPUÿ24. After CPUÿ24 reads the count and increments it (Steps 226, 228), CPU 24 writes a zero to semaphore 96, thereby freeing the countÿregister 92 for use by CPUÿ24. (Step 230).

If the count read in step 226 was zero, thereby indicating that the circular buffer was empty before CPU 24 added the last block, CPU 24 interrupts CPU 32 to notify it that the buffer now contains data. (Steps 232, 234). (The mechanism by which CPU 24 interrupts CPU 32 will be described in detail below.) If the count is greater than zero, an interrupt should already be pending due to the previously loaded data block which has not yet been removed. Accordingly, CPU 24 returns to other operations. (Step 236).

Referring to Figs. 6(a) and 6(b), CPU responds to an interrupt by first clearing the interrupt as described below. (Step 310). It proceeds to remove a block of data from memory 31 by first reading the removal pointer 90 to determine the location of the next block in the buffer. (Step 311). CPU 32 next initializes registers 56, 58 with the address of the first location in the block to be emptied and sets the read increment bit RI2 in register 56. (Step 312). It then empties the block by successively reading from memory 31 and writing the data to cache 33. (Step 314). When the entire block is emptied, CPU 32 updates the removal pointer 90 to point to the next block in the buffer. (Step 316).

Finally, CPU 32 updates the count register 92 to reflect the removal of a block from the buffer. Toward this end, it first reads the count semaphore 96. (Step 318) If the semaphore is set to one, indicating that CPU 24 has control of the count, CPU 32 repeatedly reads the semaphore until it returns to a zero. (Step 320) Once the semaphore clears, CPU 32 reads the count from register 92 and decrements register 92 to reflect the removal of a block. (Steps 322, 324). After decrementing the count, CPU 32 releases the count register 92 by clearing the semaphore 96. (Step 326).

CPU 32 then examines the updated count to determine if the buffer is empty. (Step 328). If the

buffer contains another data block, CPU 32 continues to read blocks until the buffer is emptied. (Step 330). Once the buffer is empty, CPU 32 returns to other tasks until a new interrupt appears of bus 62 indicating that new data has been loaded into the data buffer. (Step 332).

The control message blocks 74, 76 operate in essentially the same manner to transfer control messages. More specifically, control register block 72 includes an entry pointer 98 indicating the location of the next available message container, and a removal pointer 100 indicating the location of the next message container to be emptied. (Fig. 4). It further includes a count register 102 indicating the number of control messages stored in the message block 74. Semaphore block 70 includes an associated semaphore 104 for use in arbitrating competing requests for access the count register 102.

Similarly, control register block 72 includes a removal pointer 105, an entry pointer 106, and a count register 107, for the use in controlling the passing of messages through control message block 76. Semaphore block 70 includes a semaphore 108 for arbitrating between competing requests for access to count register 107.

INTERRUPTS

As explained above, once CPU 24 loads the first data or message block to memory 31, (that is memory 31 was previously empty), it notifies CPU 32 that the block is available by interrupting CPU 32. The following describes in more detail the mechanism for generating the interrupt.

Referring to Figs. 2 and 3, interface 30 includes a control status register 66 used by CPU 24 to generate an interrupt to CPU 32. More specifically, CPU 24 writes to register 66 to set certain bits in the

register which request interface 30 to interrupt CPU 32. To write to register 66, CPU 24 asserts an address dedicated to the register 66. Decoders 44 and 46 decode the address causing decoder 46 to assert control register access signal R4. Upon receipt of register signal R4, register 66 loads data from data bus DB into its register cells.

CPU 24 requests interface 30 to interrupt CPU 32 by setting interrupt bit IG1 of control status register 66 and loading bits zero through five (MID0-MID5) with the identification code ID1 of CPU 32. The identification code ID1 is applied to a comparator 110 which compares ID1 with an identification code "CODE'" assigned to CPU 32. The comparator output and the interrupt generation bit IG1 are applied to a three input AND gate 112, thereby requesting an interrupt.

Interface 30 includes a second control status register 68, virtually identical to register 66, which is used by CPU 32 to enable and disable interrupt requests from CPU 24 by setting or clearing an interrupt enable bit IE2. The interrupt enable bit IE2 is applied to the third input of AND gate 112. If IE2 is set, the activation of the other two inputs causes the output of AND gate 112 to become asserted, triggering an interrupt latch 114 to set. Once set, interrupt latch 114 asserts an interrupt signal "Interrupt2" on bus 62. CPU 32 clears the interrupt by setting the interrupt pending bit IP2 in its status register 68, thereby causing latch 114 to clear.

CPU 32 interrupts CPU 24 in the same manner. More specifically, it sets interrupt bit IG2 in register 68 and loads bits MID0 - MID5 with the identification code ID2. The bit IG1 is applied to an input of an AND gate 113. The Identification code bits ID2 are applied to an input of a comparator 111. Comparator 111 compares ID2 to "code2". If ID2 and Code2 are identical, the comparator supplies a "match"

signal to AND gate 113. Finally, interrupt enable bit IE1 from status register 66 is applied to the third input of AND gate 113. If all three inputs to AND gate 113 are asserted, the output of AND gate 113 sets an interrupt latch 115. Once set, Interrupt latch 115 asserts an interrupt signal "Interrupt1" across cable 19. Interface 28 forwards the interrupt to bus 22 thereby interrupting CPU 24. CPU 24 clears the interrupt by setting interrupt pending bit IP1 in its status register 66.

The interrupt mechanism also provides the means by which the backup fileserver determines when the primary has failed. More specifically, in normal operation, CPU 24 will regularly interrupt CPU 32 with new data to be loaded to the backup disk 34. If CPU 32 does not receive an interrupt within a specified period of time, it assumes the primary has failed and proceeds to assume responsibility for the primary.

It is possible that a properly operating primary will not send any data to the backup fileserver for a long period of time due to a lack of activity on the network. Accordingly, the primary also monitors the length of time since it last interrupted CPU 32. If the specified period of time is about to expire, CPU 24 sends an "Alive" message to control block 74 and interrupts CPU 32 to notify it that the buffer contains a message. CPU 32 will respond to the interrupt, read the Alive message and return to its normal operation. In this manner, CPU 24 notifies CPU 32 that it is operational even during moments when no data needs to be copied to backup disk 34. Similarly, CPU 32 regularly sends alive messages to CPU 24 to notify it that CPU 32 remains operational.

Referring to Figs. 7(a)-7(d), if CPU 24 fails to interrupt CPU 32 for the specified length of time, the backup fileserver assumes responsibility for the primary. The backup fileserver 16 sends a "shut down"

message to the block 76 to indicate to the primary that it is taking over. (Step 410). Backup fileserver 16 then mounts the backup filesystem as a local filesystem (Step 412) and activates its network interface 116. (Fig. 1)(Step 416). It then broadcasts an "Address Resolution Protocol" packet (herein "ARP" packet) over link 14 via network interface 116 indicating that it will now handle all traffic formerly directed to the primary. More specifically, each client node of the primary maintains a Node ID table containing the network interface address used by each node recognized by the client's operating system. The "ARP" packet instructs each client node to modify its table by replacing the interface address of network interface 20 in the primary with the address of network interface 116 in the backup. (Step 418).

After sending the ARP packet, the backup fileserver begins maintaining a record of all disk data blocks which are amended so that the primary's outdated data blocks can be replaced at a future time. More specifically, the backup fileserver allocates a block CPU memory for storage of a journal bit map. (Step 420). Each bit in the map corresponds to a single disk data block. If a given block is written to or otherwise modified, the corresponding bit in the journal bit map is set.

To update the newly created journal bit map, the backup fileserver first reads a journal file which describes all changes to the filesystem within the last fifteen seconds. (Step 422). The backup fileserver examines the journal and, for each data block modified in the last fifteen seconds, sets a corresponding bit in the journal bit map. (Step 424). The backup fileserver continues to monitor the journal file, updating the bit map with each modification of the filesystem.

The primary fileserver will eventually be restarted after the failure condition is remedied. Upon being restarted, the primary sends an "Alive" message to control message block 74. The backup fileserver reads the "Alive" message from the control message block and begins returning control to the primary. (Step 426).

Toward this end, it first deactivates its network interface 116 from receiving any more packets. (Step 428). It then waits fifteen seconds for all pending network packets to be processed. (Step 430). More specifically, the backup fileserver may have already received packets from the network which have not yet been incorporated into the filesystem. Further, it may have already begun preparing packets for transmission over the network. Accordingly, during the fifteen second waiting period, the backup fileserver updates the filesystem to reflect any required changes and transmits all pending packets.

After waiting for fifteen seconds, the backup fileserver sends a disk data block allocation bit map to the primary through the dual ported memory 31. (Step 432). Each bit of the block allocation bit map corresponds to a disk data block. A bit set to one indicates that the corresponding disk block is used, a zero indicates that the disk block is free. The backup fileserver then examines the journal bit map to determine if it reflects the most recent changes to the filesystem. (Step 434). If not, the backup fileserver updates the journal bit map to reflect the most recent changes. (Steps 436, 438).

Once the journal bit map is updated, the fileserver scans the journal bit map tc identify each disk data block which has been modified since fifteen seconds prior to the failure of the primary. (Step 440). It then sends each modified data block to the primary through dual ported memory 31 using the data

block transfer procedure described above. (Step 442). After completing this transfer, it deallocates the CPU memory block which contains the journal bit map (Step 444) and sends a "Journal Done" message to the primary through control message block 76. (Step 446).

Referring to Figs. 8(a) and 8(b), the following describes the operation of the primary file server 15 in returning to operation from a failure. The primary first sends an "alive" message to the backup fileserver through the control message block 74 of the dual ported memory 31. (Step 510). When the backup fileserver responds by sending the disk data block allocation bit map, the primary replaces its old data block allocation bit map with the newly arrived allocation bit map. (Step 512). It then proceeds to read each arriving modified data block from the dual ported interface and to write the block to its disk 26. (Step 514). When the backup fileserver sends a Journal Done message, indicating that all blocks have been sent (Step 516), the primary activates its network interface (Step 518) and mounts the filesystem (Step 520). It then broadcasts a ARP packet instructing all client nodes to amend their node Id table to indicate that the primary has returned to service (Step 522). It finally sends an "on-line" message to the backup through control message block 76, instructing the backup to return to its role as a backup (Step 524).

When the backup fileserver is first powered on, the contents of the control register block 72 are invalid (i.e., in an unknown state). Accordingly, the semaphore block 70 includes a power up semaphore 109 to provide a mechanism for notifying CPU 24 and CPU 32 that the control entries are not valid. Both CPU 32 and CPU 24 read the semaphore bus when their respective fileserver's are first powered on. Semaphore 109 is initially set to zero when the backup

fileserver is powered up to indicate that control register block 72 is not initialized. The first of CPU 24 and CPU 32 to read a zero in semaphore 109 assumes responsibility for initializing these locations. Upon being read by either CPU, the semaphore is automatically set to a one to indicate to any CPU which subsequently reads the semaphore that the first CPU has assumed responsibility for initialization.

Referring again to Fig. 1, backup fileserver 16 may operate both as a backup fileserver and as a second primary fileserver. Toward this end, fileserver 16 includes two network interfaces 116, 118. As explained above, interface 116 is used to access communication link 14 when fileserver 16 has assumed responsibility for the failed primary. Interface 118 is used by fileserver 16 to communicate over link 14 in its capacity as a second primary fileserver.

Additions, subtractions, deletions and other modifications of the preferred particular embodiments of the invention will be apparent to those practiced in the art and are within the scope of the following claims.

What is claimed is:

1. In fault tolerant network fileserver system comprising:

a network communication link,

a plurality of nodes connected to the network communication link,

a primary fileserver for storing files from said plurality of nodes connected to the network communication link; and

a backup fileserver for storing copies of files from said primary fileserver;

the improvement comprising

said primary fileserver comprising:

a primary computer processor,

a primary storage disk,

a first network interface connected to said network communication link, and

a first independent interface connected to said backup fileserver and responsive to commands from the primary processor for communicating information to said backup fileserver;

said backup fileserver comprising:

a backup computer processor,

a backup storage disk,

a backup network interface connected to said network communication link, and

a second independent interface connected to said first independent interface for receiving information from said primary fileserver and responsive to commands from said back-up processor for reading information stored in said memory, said second independent interface comprising a dual ported memory for storing information received from said primary fileserver.

2. The fileserver system of claim 1 wherein said second independent interface further comprises an interrupt means for interrupting said backup computer processor to notify said backup computer processor that said dual ported memory contains information received from said primary fileserver.

3. The fileserver system of claim 1 wherein said dual ported memory comprises:

a group of data memory blocks each memory block for storing data to be copied to said backup storage disk,

an entry pointer register for storing an entry pointer identifying a next available data block,

a removal pointer register for storing a removal pointer identifying a next data block to be emptied, and

a count register for storing the number of data memory blocks containing data.

4. The fileserver system of claim 3 wherein said dual ported memory further comprises:

a first control message block for storing control messages from said primary fileserver to said backup fileserver, and

a second control message block for storing control messages from said backup fileserver to said primary fileserver.

5. The fileserver system of claim 3 wherein said second independent interface further comprises means responsive to signals from said first independent interface for modifying the contents of a common register assessable to the backup computer processor to indicate the entry of data into a memory block, and wherein said second independent interface is further

responsive to signals from the backup computer processor
for modifying said common register to indicate the removal of data from a memory block, said dual ported memory comprising a semaphore register for arbitrating access to said common register.

6. The fileserver system of claim 5 wherein said semaphore register has a first and a second state of operation; whereby when read by a processor while in said first state, the semaphore register provides a register available code and automatically enters said second state; when read by a processor while in said second state, said semaphore register provides a register unavailable code and remains in said second state; and when written to by a processor while in said second state, said semaphore register returns to said first state.

7. The fileserver system of claim 6 wherein said common register is said count register, and wherein said primary fileserver has means for incrementing said count register to indicate the entry of data in a data memory block and said backup processor has means for decrementing said count register to indicate the removal of data from a data memory block.

8. The fileserver system of claim 1 wherein said dual ported memory comprises at least one semaphore bit having a first and a second state of operation; whereby said at least one semaphore bit is in said first state when said second independent interface is powered on; and said at least one semaphore bit automatically enters said second state when read by a processor.

9. The fileserver system of claim 1 wherein said primary fileserver further comprises:

a disk operating system for controlling the copying of files received by said first network interface to said primary storage disk and to said dual ported memory, said disk operating system including means for receiving secure disk write instructions and unsecure disk write instructions directing said operating system to write specified files to said primary storage disk, said secure write instruction requesting a write to said primary storage disk more promptly than said unsecure write instruction,

said operating system responding to both secure and unsecure instructions by promptly writing said specified files to said dual ported memory, and by writing said specified files to said primary disk storage device using an unsecure write procedure.

10. The fileserver system of claim 8 wherein said disk operating system is a modified Unix operating system which, in response to synchronous disk write instructions, promptly writes said specified files to said dual ported memory, and writes said specified files to said primary storage disk using a delayed write procedure.

11. A method for maintaining on a backup network node, a backup filesystem which mirrors a primary filesystem of a primary network node, the method comprising the steps of:

receiving, at said primary node, secure disk write instructions directing said primary node to write other specified files to a primary nonvolatile storage device using a secure write procedure;

receiving, at said primary node, unsecure disk write instructions directing said primary node to write

specified files to a primary nonvolatile storage device using an unsecure write procedure, said secure write procedure writing said specified files to said primary nonvolatile storage device more promptly than said unsecure write procedure; and

in response to either of said secure and unsecure instructions,

copying said specified files to a shared memory within said backup node, and

writing said specified files to said nonvolatile storage device using said unsecure write procedure.

12. The method of claim 11 wherein writing a specified file to said shared memory comprises the steps of:

reading from an entry pointer register within said shared memory, an entry pointer identifying a next available data block portion of said shared memory, and

writing said specified file to said next available data block; and wherein said method further comprises the steps of:

reading from a removal pointer register within said shared memory, a removal pointer identifying a next data block portion of said shared memory to be emptied, and

copying said next data block portion to a storage device for storing said backup filesystem.

13. The method of claim 12 wherein writing a specified file to said shared memory further comprises the steps of:

reading from a count register within said shared memory, a count identifying the number of data blocks within said data block portion of said shared memory which contain specified files, and

incrementing said count register to indicate said writing of a specified file to a next available data

block; and wherein said method further comprises the step of:

decrementing said count register to indicate said reading of a specified file from a next data block portion to be emptied.

14. The method of claim 13 wherein said method further comprises the steps of:

reading from a semaphore register within said shared memory, an available code indicating whether said count register is available, and

resetting said semaphore register after said incrementing or decrementing of said count register to indicate that said count register is available.

15. The method of claim 12 further comprising the steps of:

reading from a power up semaphore register within said shared memory, an initialization code indicating whether selected registers within said shared memory have been initialized since said shared memory was powered on, and

initializing said selected registers if said initialization code indicates that said selected registers have not been initialized.

16. The method of claim 11 further comprising the step of:

interrupting a backup processor within said backup node to notify said backup node that said primary node is operational.

17. The method of claim 16 wherein interrupting said backup processor comprises the steps of:

writing an alive control message to a control message block within said shared memory, and

interrupting said backup processor to notify said processor that said shared memory includes a control message.

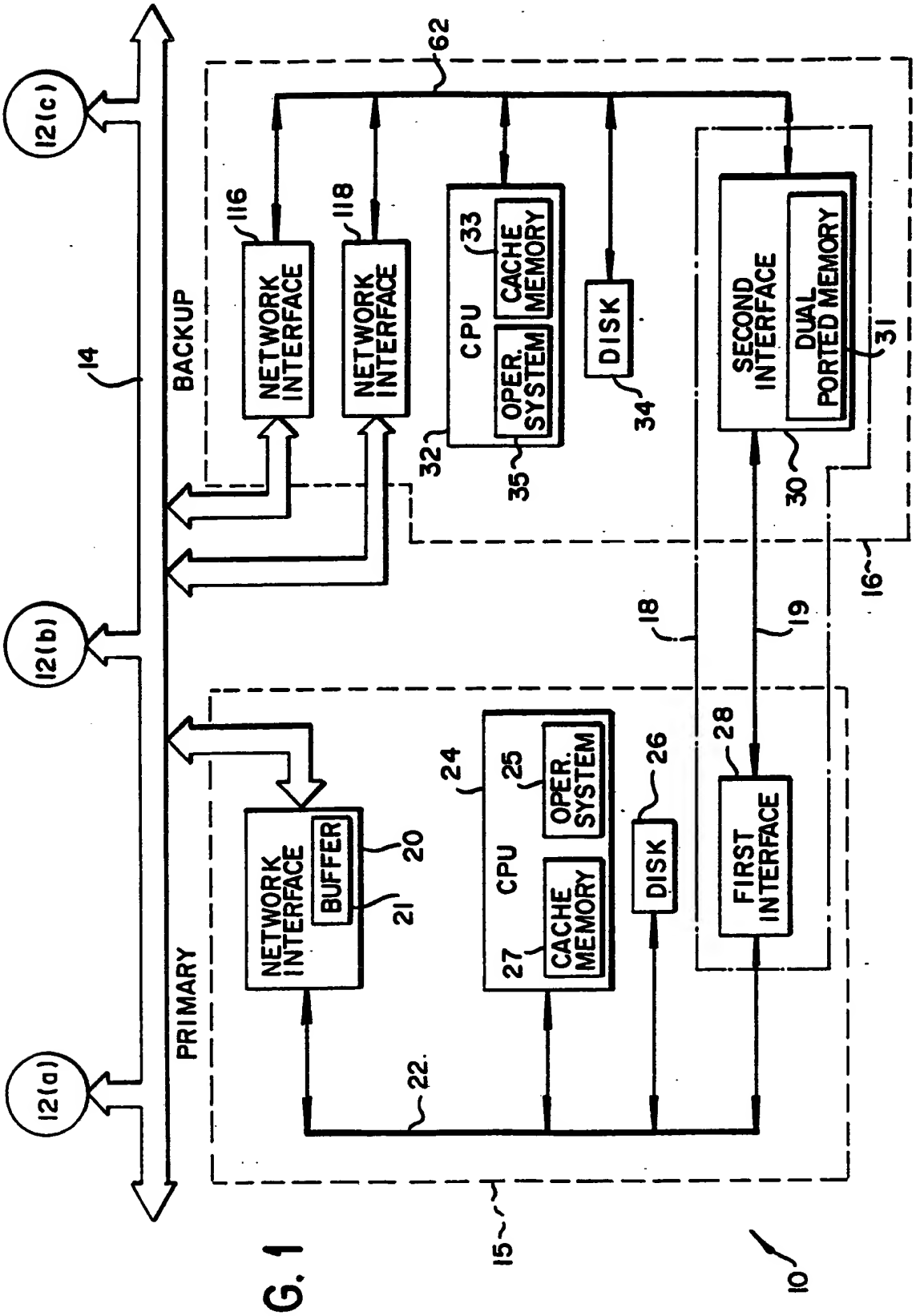

12(c)
12(b)
12(a)
14
BACKUP
PRIMARY
62
116
118
NETWORK INTERFACE
NETWORK INTERFACE
CPU
33
CACHE MEMORY
OPER. SYSTEM
35
32
DISK
34
SECOND INTERFACE
DUAL PORTED MEMORY
31
30
16
18
19
NETWORK INTERFACE
BUFFER
20
21
24
CPU
25
OPER. SYSTEM
27
CACHE MEMORY
26
DISK
28
FIRST INTERFACE
22
15
10

FIG. 1

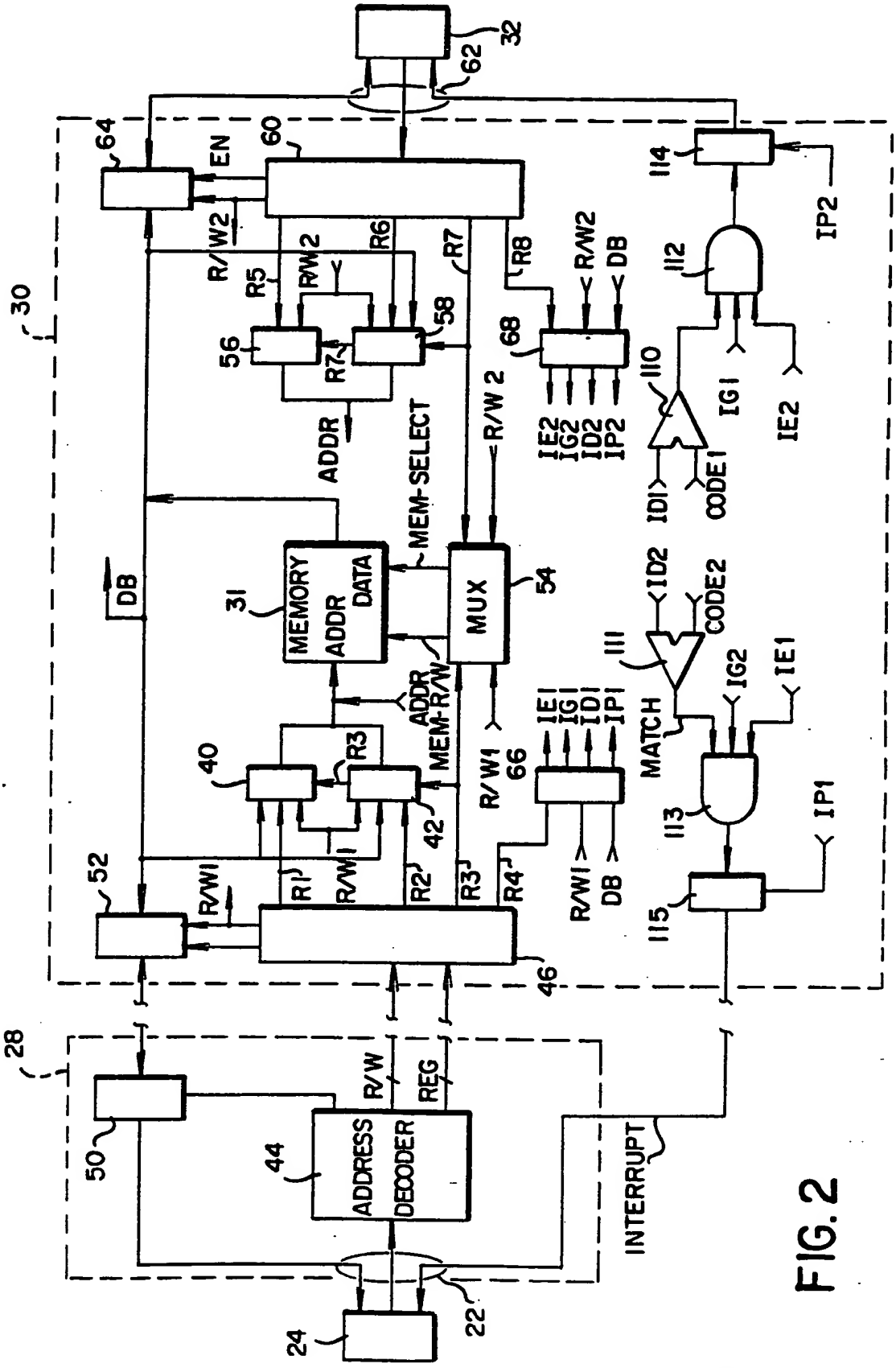
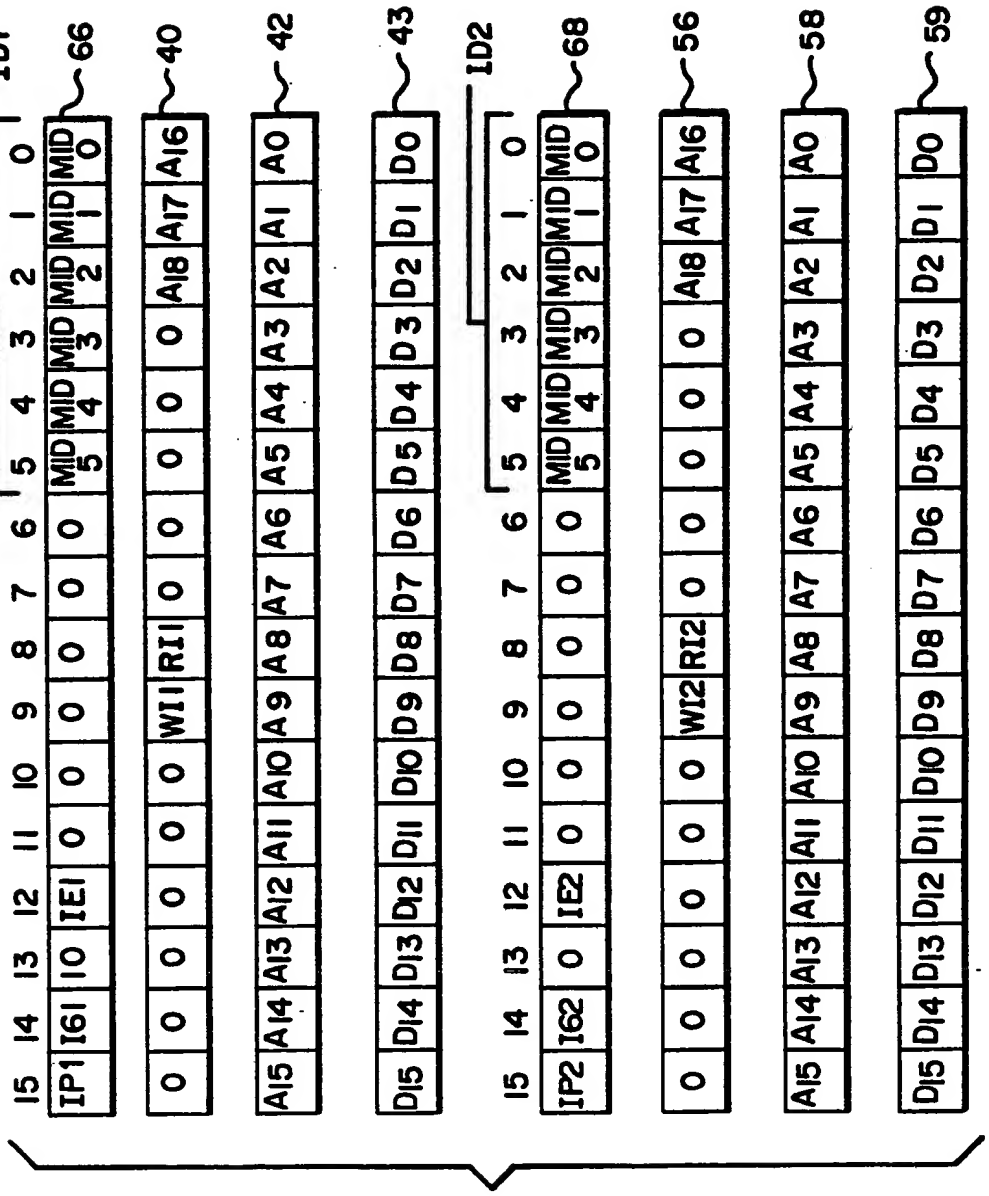

32
62
64
EN
60
R/W2
R5
R/W2
R6
R7
R8
R/W2
DB
114
IP2
112
30
56
R7
58
68
110
IG1
IE2
ADDR
MEM-SELECT
R/W 2
IE2
IG2
ID2
IP2
ID1
CODE1
DB
31
MEMORY
ADDR
DATA
MUX
54
ID2
CODE2
111
IE1
IG2
ADDR
MEM-R/W
IE1
IG1
ID1
IP1
MATCH
40
R3
42
R/W1
66
113
IP1
52
R/W1
R1
R/W
R2
R3
R4
R/W1
DB
115
46
28
R/W
REG
50
44
ADDRESS
DECODER
INTERRUPT
24
22

FIG. 2

FIG. 3


ID1
66
40
42
43
ID2
68
56
58
59
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
IP1 I61 I0 IE1 0 0 0 0 0 MID MID MID MID MID MID
0 0 0 0 0 0 WI1 RI1 0 0 0 0 0 A18 A17 A16
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
IP2 I62 0 IE2 0 0 0 0 0 MID MID MID MID MID MID
0 0 0 0 0 0 WI2 RI2 0 0 0 0 0 A18 A17 A16
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

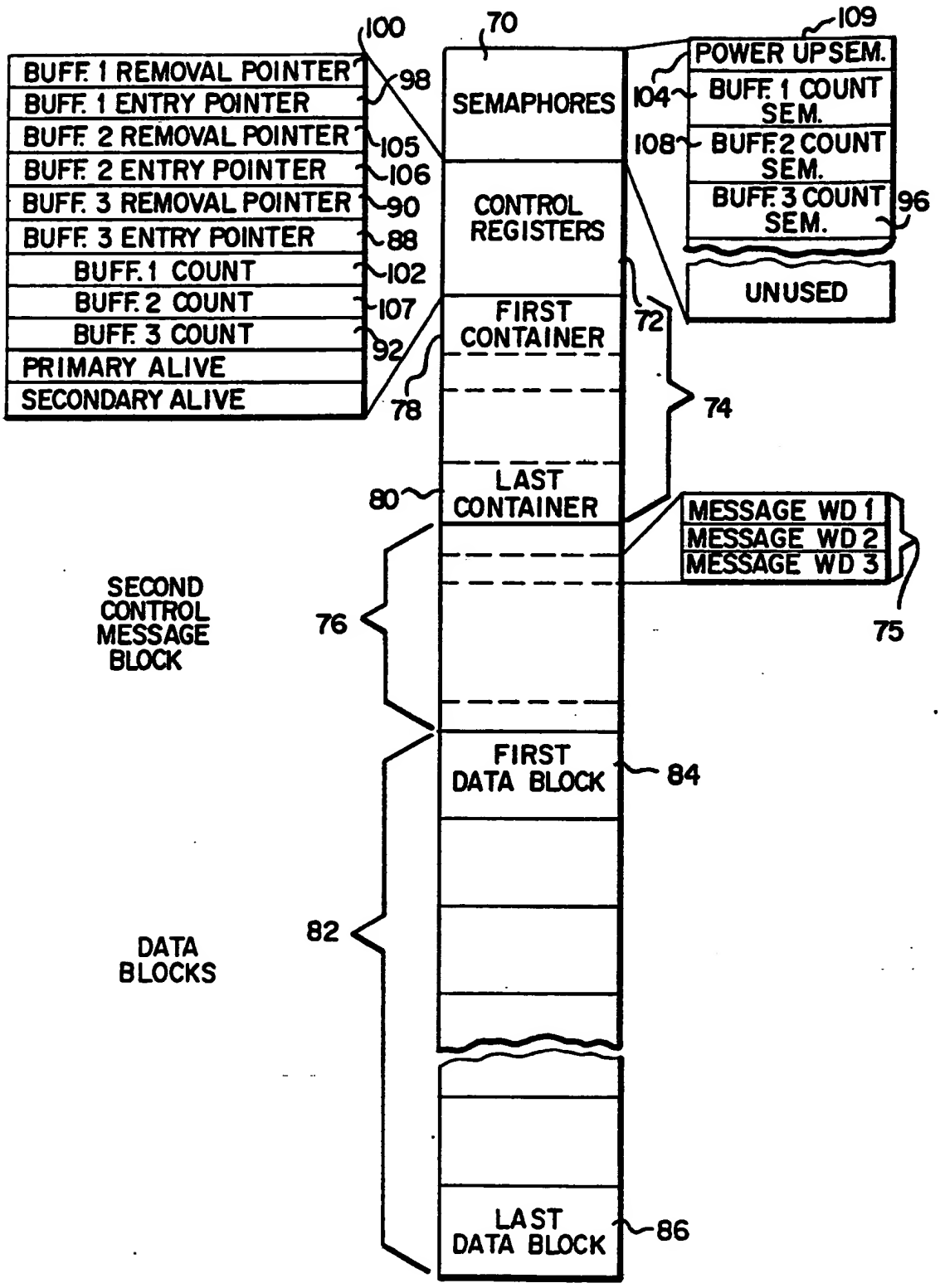

100
70
109
BUFF. 1 REMOVAL POINTER
BUFF. 1 ENTRY POINTER
BUFF. 2 REMOVAL POINTER
BUFF. 2 ENTRY POINTER
BUFF. 3 REMOVAL POINTER
BUFF. 3 ENTRY POINTER
BUFF. 1 COUNT
BUFF. 2 COUNT
BUFF. 3 COUNT
PRIMARY ALIVE
SECONDARY ALIVE
98
105
106
90
88
102
107
92
SEMAPHORES
CONTROL REGISTERS
FIRST CONTAINER
LAST CONTAINER
78
80
72
74
POWER UP SEM.
BUFF. 1 COUNT SEM.
BUFF. 2 COUNT SEM.
BUFF. 3 COUNT SEM.
UNUSED
104
108
96
MESSAGE WD 1
MESSAGE WD 2
MESSAGE WD 3
75
SECOND CONTROL MESSAGE BLOCK
76
FIRST DATA BLOCK
84
DATA BLOCKS
82
LAST DATA BLOCK
86

FIG. 4

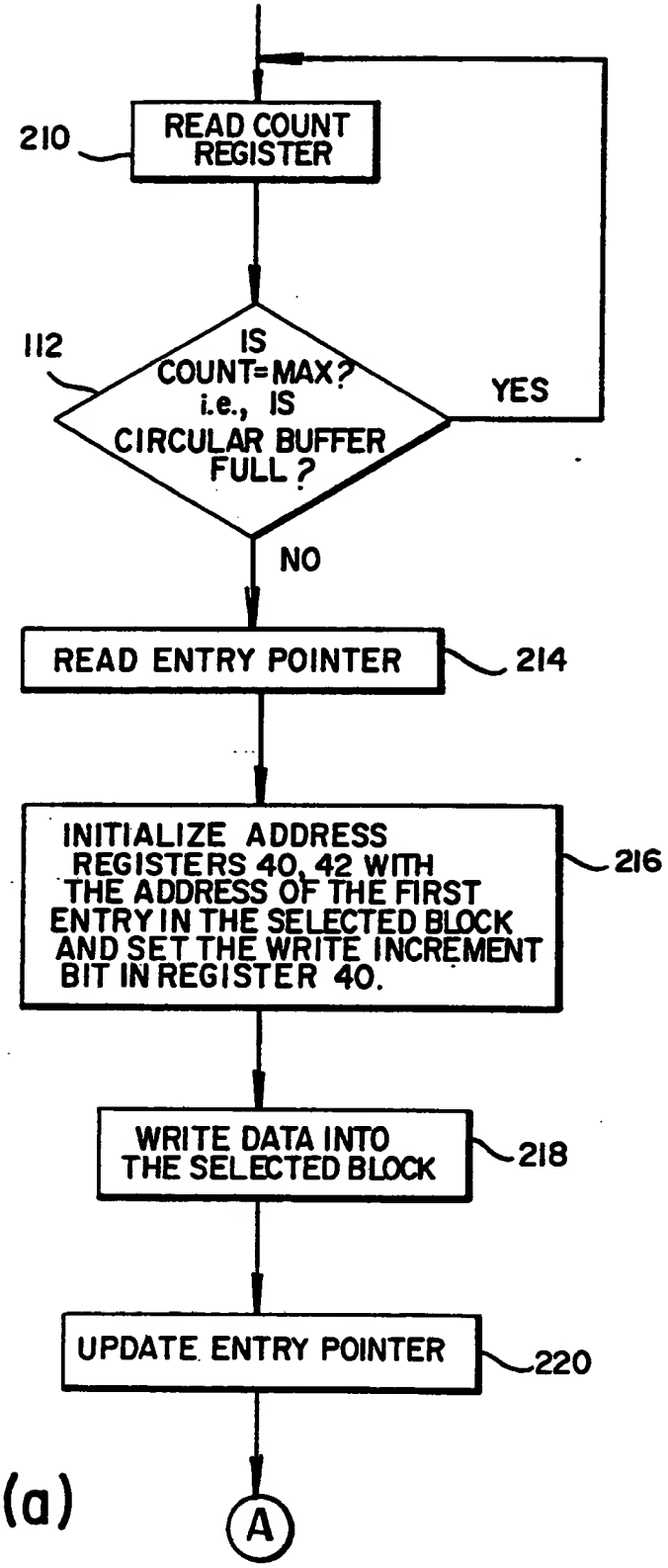

210
READ COUNT REGISTER
112
IS COUNT=MAX? i.e., IS CIRCULAR BUFFER FULL?
YES
NO
READ ENTRY POINTER
214
INITIALIZE ADDRESS REGISTERS 40, 42 WITH THE ADDRESS OF THE FIRST ENTRY IN THE SELECTED BLOCK AND SET THE WRITE INCREMENT BIT IN REGISTER 40.
216
WRITE DATA INTO THE SELECTED BLOCK
218
UPDATE ENTRY POINTER
220
A

FIG. 5(a)

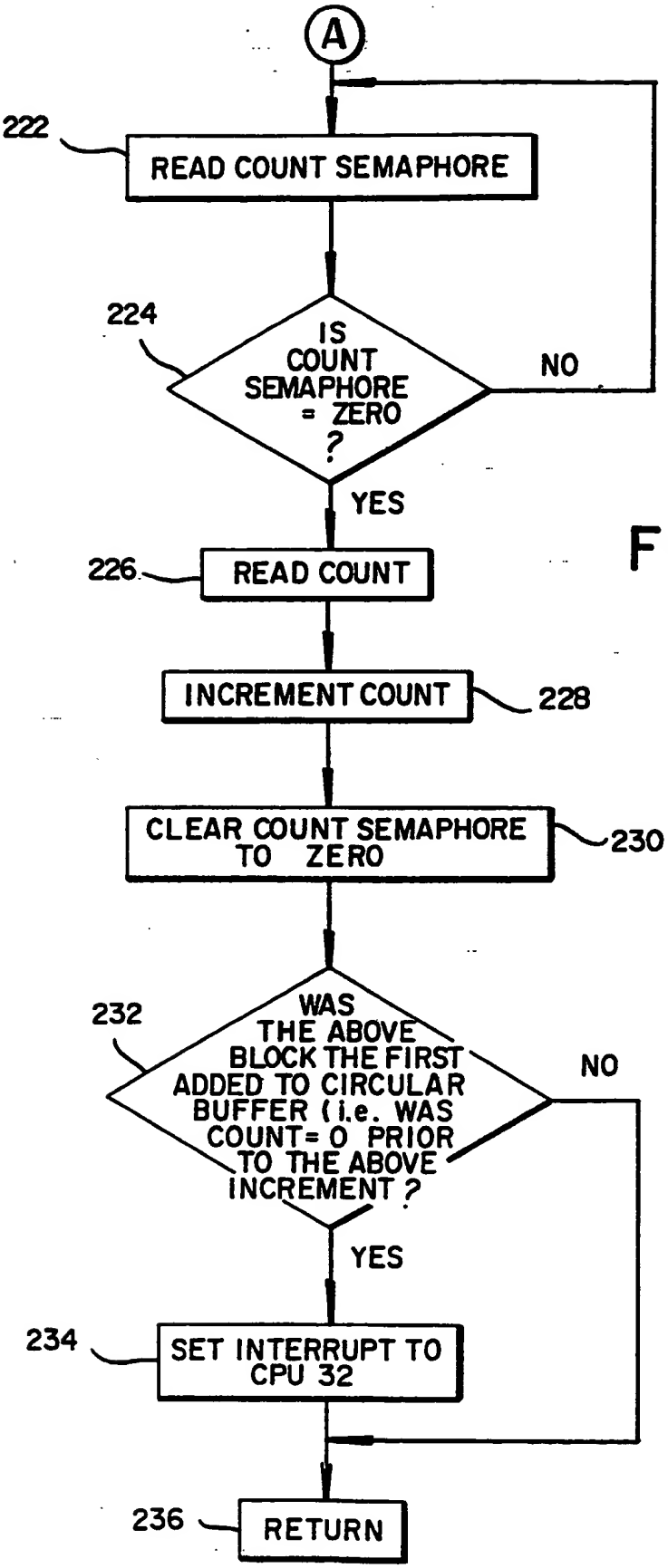

A
222
READ COUNT SEMAPHORE
224
IS COUNT SEMAPHORE = ZERO ?
NO
YES
226
READ COUNT
INCREMENT COUNT
228
CLEAR COUNT SEMAPHORE TO ZERO
230
232
WAS THE ABOVE BLOCK THE FIRST ADDED TO CIRCULAR BUFFER (i.e. WAS COUNT= 0 PRIOR TO THE ABOVE INCREMENT ?
NO
YES
234
SET INTERRUPT TO CPU 32
236
RETURN

FIG. 5(b)

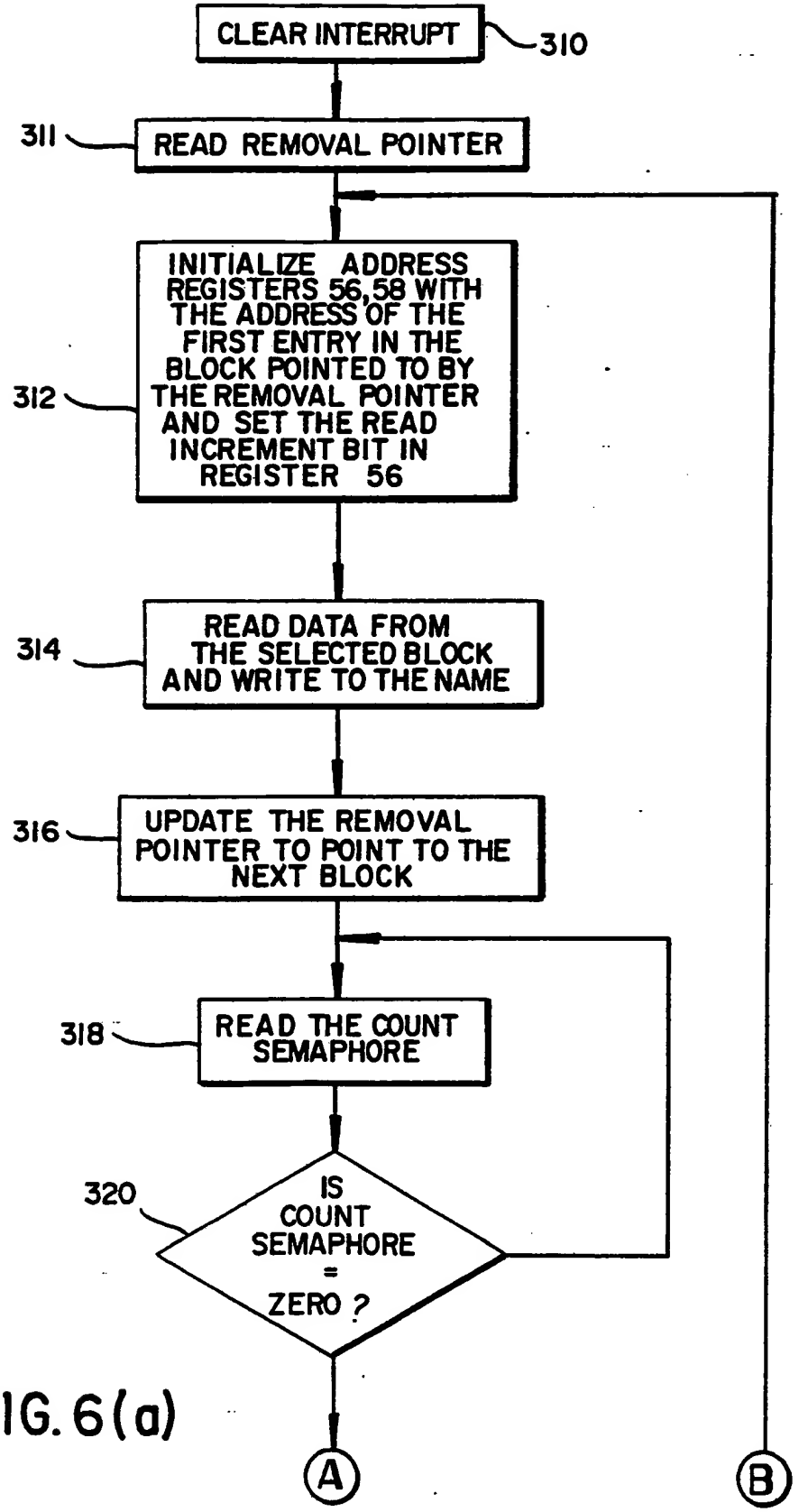

CLEAR INTERRUPT
310
311
READ REMOVAL POINTER
INITIALIZE ADDRESS REGISTERS 56,58 WITH THE ADDRESS OF THE FIRST ENTRY IN THE BLOCK POINTED TO BY THE REMOVAL POINTER AND SET THE READ INCREMENT BIT IN REGISTER 56
312
READ DATA FROM THE SELECTED BLOCK AND WRITE TO THE NAME
314
UPDATE THE REMOVAL POINTER TO POINT TO THE NEXT BLOCK
316
READ THE COUNT SEMAPHORE
318
IS COUNT SEMAPHORE = ZERO ?
320
A
B

FIG. 6(a)

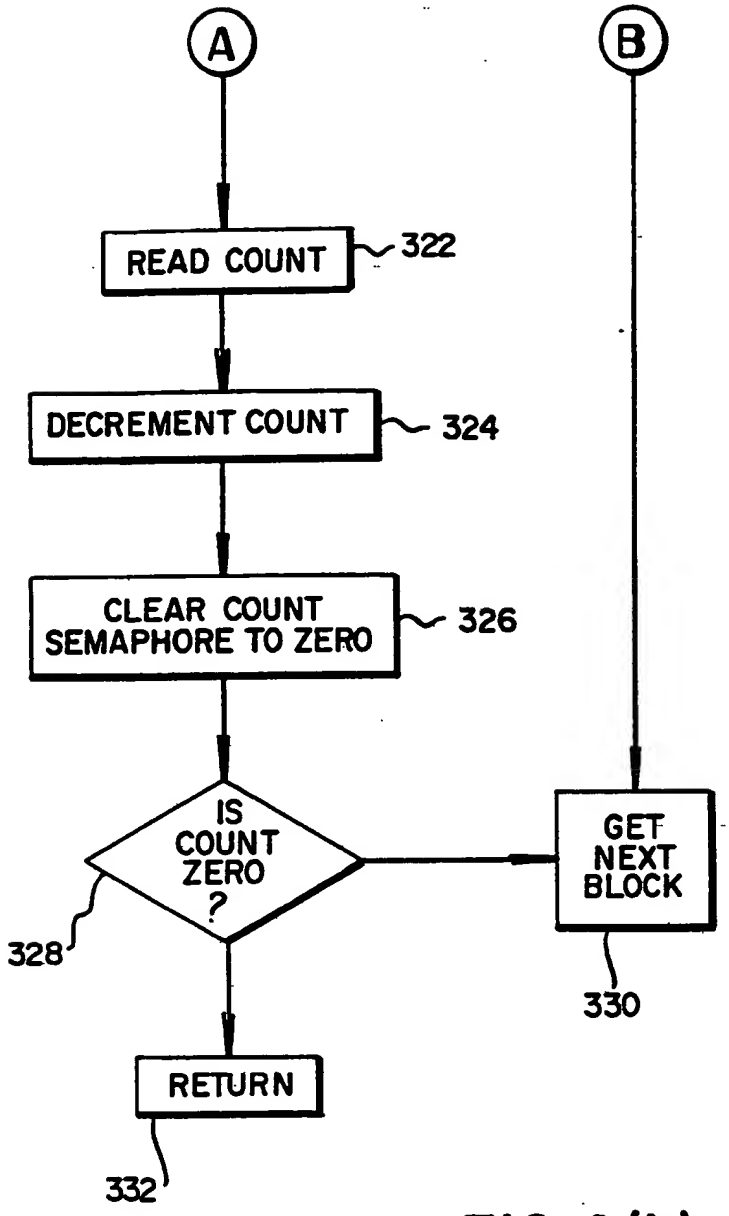

A
B
READ COUNT
322
DECREMENT COUNT
324
CLEAR COUNT SEMAPHORE TO ZERO
326
IS COUNT ZERO ?
328
GET NEXT BLOCK
330
RETURN
332

FIG. 6(b)

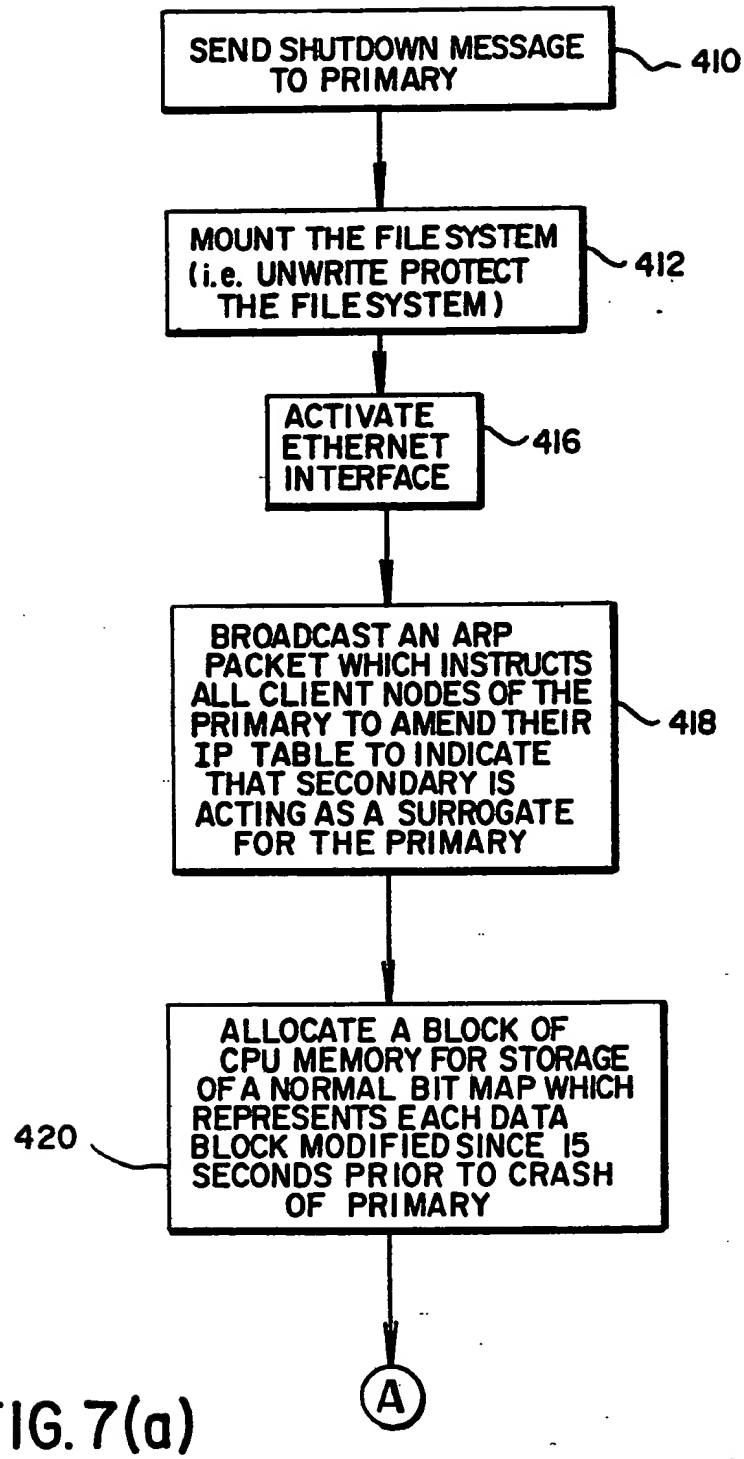

SEND SHUTDOWN MESSAGE TO PRIMARY
410
MOUNT THE FILE SYSTEM (i.e. UNWRITE PROTECT THE FILE SYSTEM)
412
ACTIVATE ETHERNET INTERFACE
416
BROADCAST AN ARP PACKET WHICH INSTRUCTS ALL CLIENT NODES OF THE PRIMARY TO AMEND THEIR IP TABLE TO INDICATE THAT SECONDARY IS ACTING AS A SURROGATE FOR THE PRIMARY
418
ALLOCATE A BLOCK OF CPU MEMORY FOR STORAGE OF A NORMAL BIT MAP WHICH REPRESENTS EACH DATA BLOCK MODIFIED SINCE 15 SECONDS PRIOR TO CRASH OF PRIMARY
420
A

FIG. 7(a)

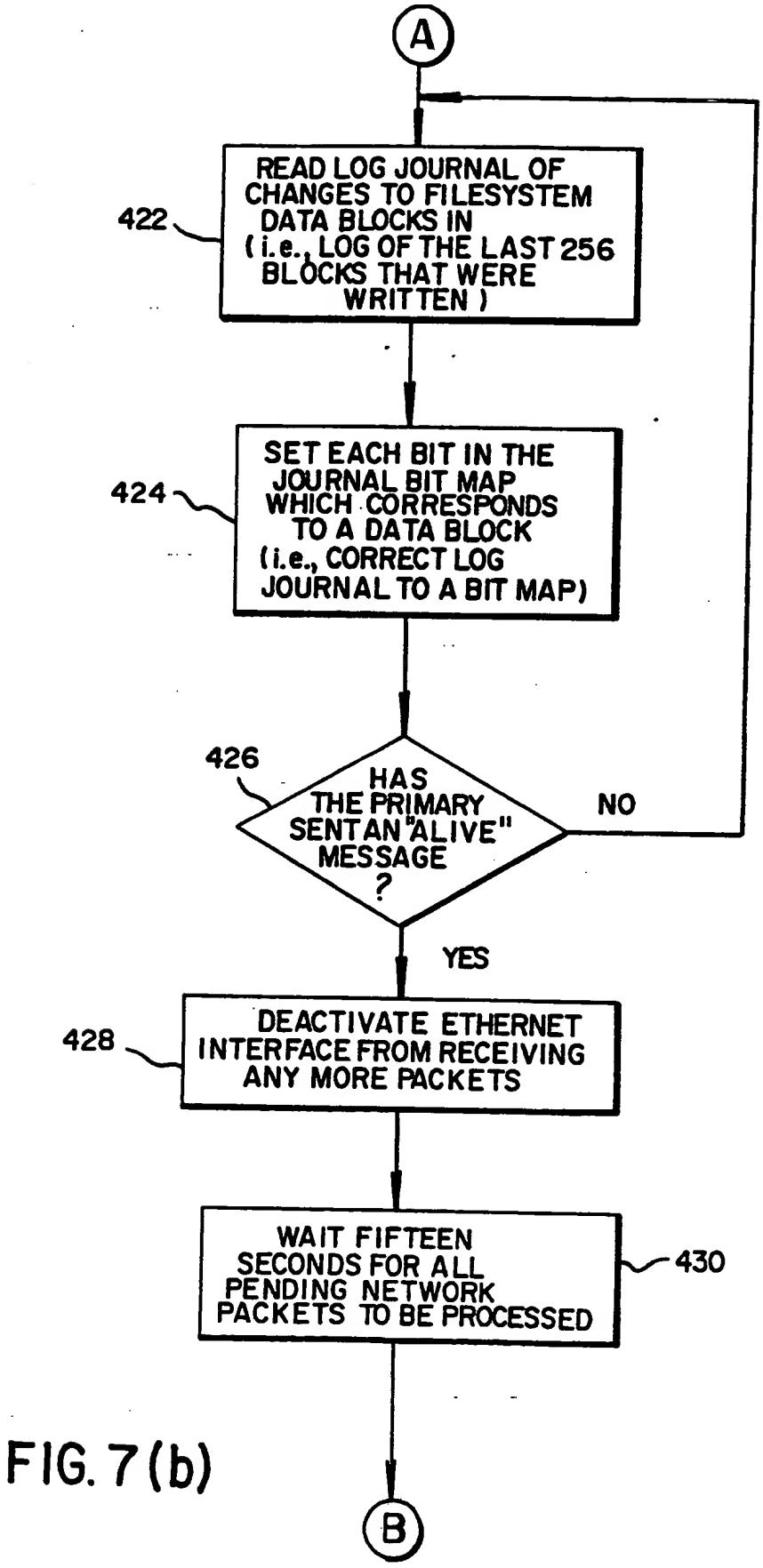

A
READ LOG JOURNAL OF CHANGES TO FILESYSTEM DATA BLOCKS IN (i.e., LOG OF THE LAST 256 BLOCKS THAT WERE WRITTEN)
422
SET EACH BIT IN THE JOURNAL BIT MAP WHICH CORRESPONDS TO A DATA BLOCK (i.e., CORRECT LOG JOURNAL TO A BIT MAP)
424
426
HAS THE PRIMARY SENT AN "ALIVE" MESSAGE ?
NO
YES
DEACTIVATE ETHERNET INTERFACE FROM RECEIVING ANY MORE PACKETS
428
WAIT FIFTEEN SECONDS FOR ALL PENDING NETWORK PACKETS TO BE PROCESSED
430
B

FIG. 7(b)

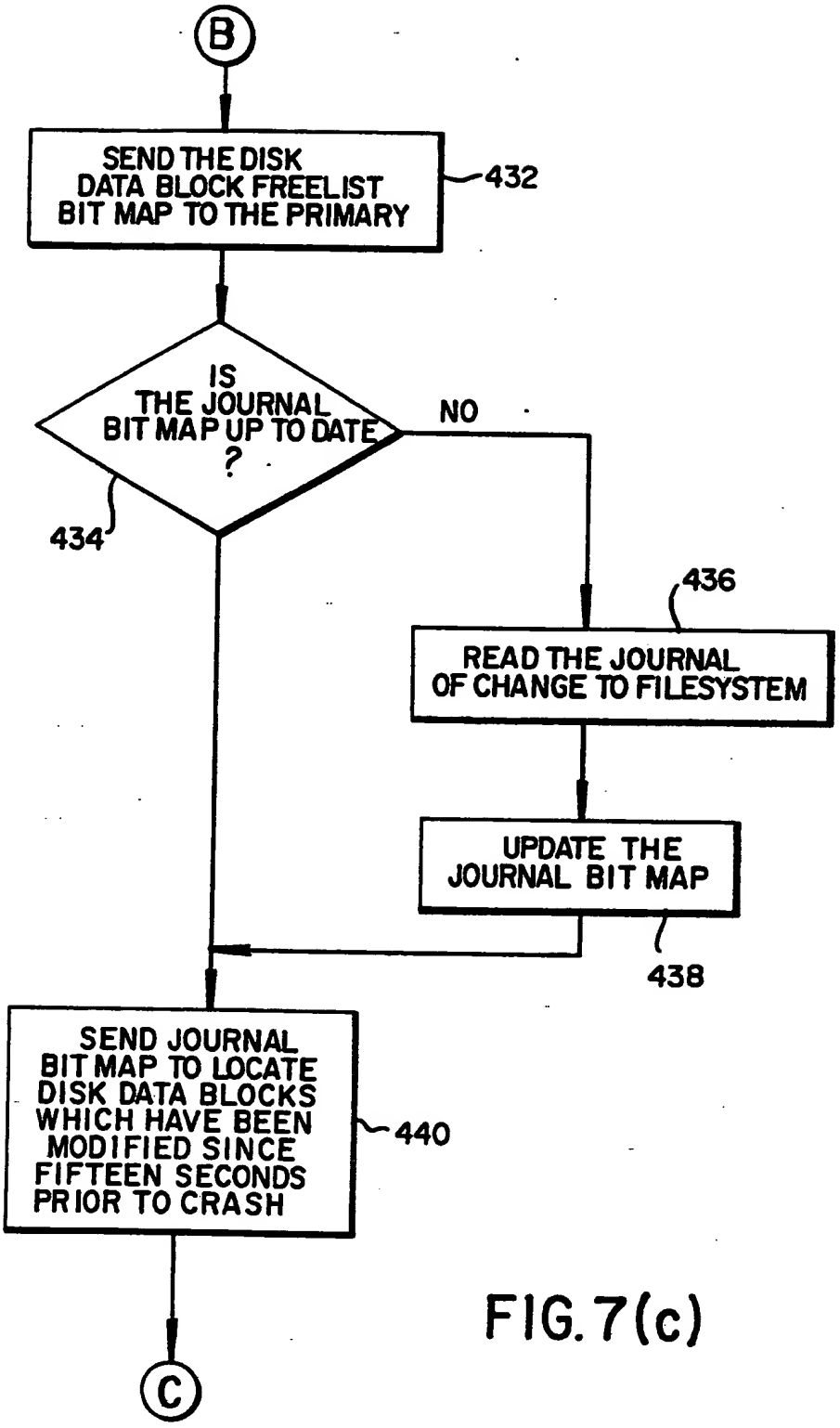

B
SEND THE DISK DATA BLOCK FREELIST BIT MAP TO THE PRIMARY
432
IS THE JOURNAL BIT MAP UP TO DATE ?
NO
434
436
READ THE JOURNAL OF CHANGE TO FILESYSTEM
UPDATE THE JOURNAL BIT MAP
438
SEND JOURNAL BIT MAP TO LOCATE DISK DATA BLOCKS WHICH HAVE BEEN MODIFIED SINCE FIFTEEN SECONDS PRIOR TO CRASH
440
C

FIG. 7(c)

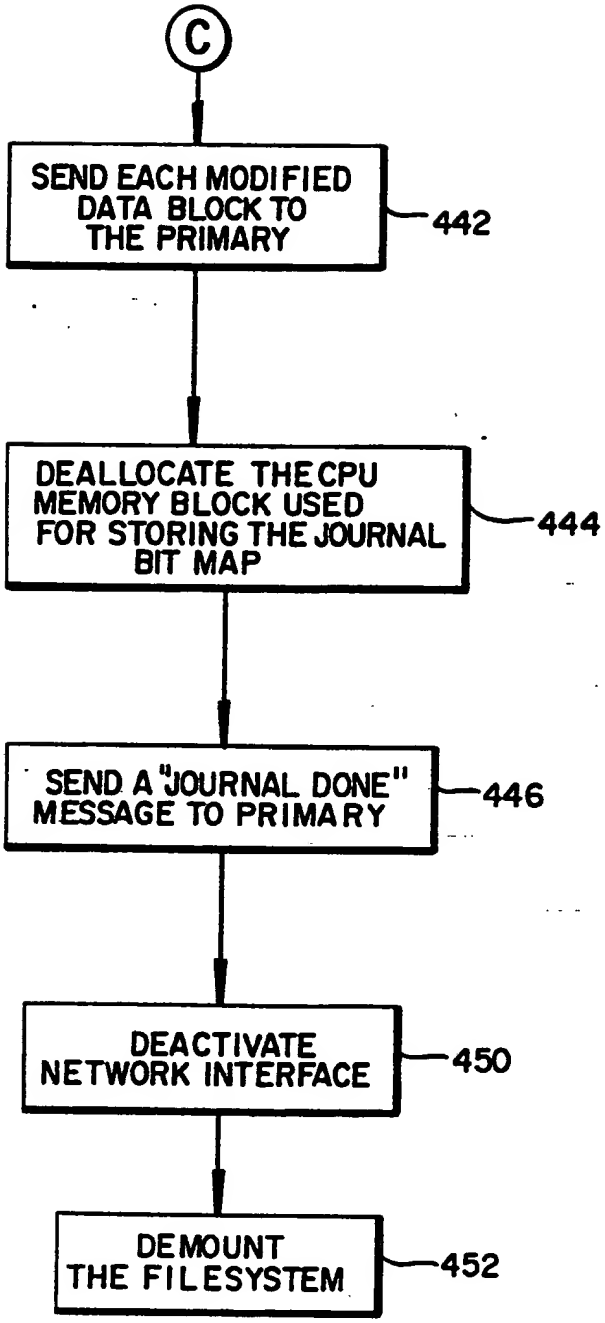

C
SEND EACH MODIFIED DATA BLOCK TO THE PRIMARY
442
DEALLOCATE THE CPU MEMORY BLOCK USED FOR STORING THE JOURNAL BIT MAP
444
SEND A "JOURNAL DONE" MESSAGE TO PRIMARY
446
DEACTIVATE NETWORK INTERFACE
450
DEMOUNT THE FILESYSTEM
452

FIG. 7(d)

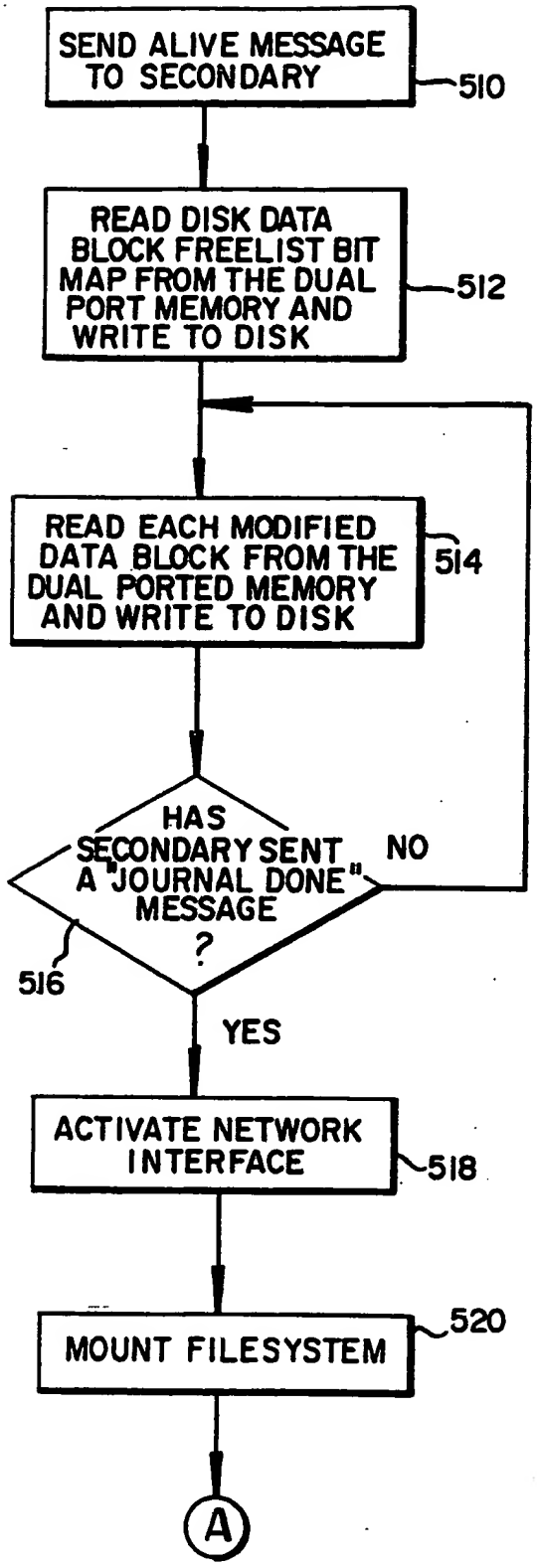

SEND ALIVE MESSAGE TO SECONDARY
510
READ DISK DATA BLOCK FREELIST BIT MAP FROM THE DUAL PORT MEMORY AND WRITE TO DISK
512
READ EACH MODIFIED DATA BLOCK FROM THE DUAL PORTED MEMORY AND WRITE TO DISK
514
HAS SECONDARY SENT A "JOURNAL DONE" MESSAGE ?
NO
516
YES
ACTIVATE NETWORK INTERFACE
518
MOUNT FILESYSTEM
520
A

FIG. 8(a)

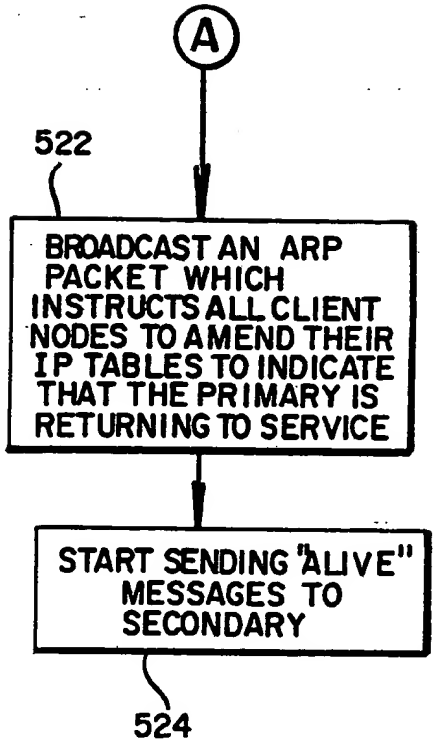

A
522
BROADCAST AN ARP PACKET WHICH INSTRUCTS ALL CLIENT NODES TO AMEND THEIR IP TABLES TO INDICATE THAT THE PRIMARY IS RETURNING TO SERVICE
START SENDING "ALIVE" MESSAGES TO SECONDARY
524

FIG. 8(b)

INTERNATIONAL SEARCH REPORT

International Application No PCT/US 92/03001

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G06F11/20; G06F11/14

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G06F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category[10]	Citation of Document,[11] with indication, where appropriate, of the relevant passages[12]	Relevant to Claim No.[13]
A	US,A,4 958 270 (P. F. MACLAUGHLIN, P. H. MODY) 18 September 1990 see column 2, line 54 - column 5, line 28 see column 6, line 66 - column 7, line 10 see figures 1,3,4	1,3,4,9, 11
A	EP,A,0 359 471 (COMPAQ COMPUTER CORP.) 21 March 1990 see column 1, line 10 - column 3, line 15 see figures 1-3	1,9,11
A	WO,A,8 909 452 (NCR CORP.) 5 October 1989 see page 4, line 19 - page 5, line 14 see page 6, line 22 - page 7 see page 8, line 17 - page 10 see figures 1-4	1,3,5,11

-/--

° Special categories of cited documents :[10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search	Date of Mailing of this International Search Report
13 JULY 1992	20 JUIL. 1992
International Searching Authority	**Signature of Authorized Officer**
EUROPEAN PATENT OFFICE	JOHANSSON U.C.

Form PCT/ISA/210 (second sheet) (January 1985)

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category °	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.
A	ACM TRANSACTIONS ON COMPUTER SYSTEMS vol. 7, no. 1, February 1989, NEW YORK US pages 1 - 24; A. BORG ET AL: 'Fault-Tolerance Under UNIX' see page 3, line 26 - line 44 see page 12, line 21 - page 14, line 8	1,4,9, 10,11

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. US 9203001 SA 59094

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report. The members are as contained in the European Patent Office EDP file on 13/07/92
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
US-A-4958270	18-09-90	EP-A-	0460308	11-12-91
EP-A-0359471	21-03-90	JP-A-	2135550	24-05-90
WO-A-8909452	05-10-89	JP-A-	1253061	09-10-89
		EP-A-	0377684	18-07-90